Generation of feasible deployment configuration alternatives for Data Distribution Service based systems

Tekinerdogan, B., Çelik, T., & Köksal, Ö.

This is a "Post-Print" accepted manuscript, which has been published in "Computer Standards and Interfaces"

This version is distributed under a non-commercial no derivatives Creative Commons

([CC-BY-NC-ND](#)) user license, which permits use, distribution, and reproduction in any medium, provided the original work is properly cited and not used for commercial purposes. Further, the restriction applies that if you remix, transform, or build upon the material, you may not distribute the modified material.

Please cite this publication as follows:

Tekinerdogan, B., Çelik, T., & Köksal, Ö. (2018). Generation of feasible deployment configuration alternatives for Data Distribution Service based systems. *Computer Standards and Interfaces*, 58, 126-145. DOI: 10.1016/j.csi.2018.01.002

You can download the published version at:

<https://doi.org/10.1016/j.csi.2018.01.002>

Generation of Feasible Deployment Configuration Alternatives for Data Distribution Service Based Systems

BEDİR TEKİNERDOĞAN, Wageningen University

TURGAY ÇELİK, OPSGENIE

ÖMER KÖKSAL, Wageningen University

Data Distribution Service (DDS) has been defined by the OMG to provide a standard data-centric publish-subscribe programming model and specification for distributed systems. DDS has been applied for the development of high performance distributed systems such as in the defense, finance, automotive, and simulation domains. To support the analysis and design of a DDS-based distributed system, the OMG has proposed the DDS UML Profile. A DDS-based system usually consists of multiple participant applications each of which has different responsibilities in the system. These participants can be allocated in different ways to the available resources, which leads to different configuration alternatives. Usually, each configuration alternative will perform differently with respect to the execution and communication cost of the overall system. In general, the deployment configuration is selected manually based on expert knowledge. This approach is suitable for small to medium scale applications but for larger applications this is not tractable. In this paper, we provide a systematic approach for deriving feasible deployment alternatives based on the system design and the available physical resources. The system design is supported by the DDS UML profile that we have extended to support the generation of feasible deployment alternatives. Based on the modeled system design and physical resources, the feasible deployment alternatives can be algorithmically derived and automatically generated using the developed tools. We illustrate the approach for deriving feasible deployment alternatives of smart city parking system.

Key Words: Data Distribution Service (DDS), Software Architecture Analysis, Design Optimization, Model-Driven Development, Feasible Deployment

1. INTRODUCTION

Distributed systems realize the distributed execution of software systems over multiple resources to meet different requirements and quality factors such as performance, interoperability, multi user support. To reduce the effort for developing distributed systems, common architectures have been introduced including OMG Common Object Request Broker Architecture (CORBA) [27], Java Message Service (JMS) [28], and OMG Data Distribution Service (DDS)[26]. These middleware architectures provide common services such as name and directory services, discovery, data exchange, synchronization, transaction services, etc.

Data Distribution Service (DDS) has been defined by the OMG to provide a standard data-centric publish-subscribe programming model and specification for distributed systems. DDS has been applied for the development of high performance distributed systems such as in the defense, finance, automotive, and simulation domains. A DDS-based system usually consists of multiple participant applications each of which has different responsibilities in the system. These participants can be allocated in different ways to the available resources, which leads to different configuration alternatives. Usually, each configuration alternative will perform differently with respect to the execution and communication cost of the overall system. In general, the deployment configuration is selected manually which is suitable for small to medium scale applications but for larger applications this is not tractable. The OMG DDS specification does not provide an explicit approach to guide the distribution and allocation of the participants to optimize the deployment configuration with respect to performance. The deployment configuration is usually selected manually which is suitable for small to medium scale applications but gets intractable when larger applications are considered.

In this paper, we provide a systematic approach for deriving feasible deployment alternatives based on the system design and the available physical resources. In the approach, first the corresponding DDS-based system and the available physical resources are designed. The system design is supported by the DDS UML profile that we have extended to support the generation of feasible deployment alternatives. The design is used to define alternative execution configurations that refine the number and parameters of the corresponding design elements. Based on the application design and the execution configuration, the feasible

deployment alternatives can be algorithmically derived. The presented approach is supported by corresponding tools that support the application design, the execution configuration definition and the automatic generation of feasible deployment alternatives using model-driven development techniques. We illustrate the approach for deriving feasible deployment alternatives of smart city parking system.

The remainder of the paper is organized as follows. In section 2, we provide the background on architecture of DDS and designing DDS based systems. Section 3 defines the case study that will be used in subsequent sections. Section 4 describes the problem statement. Section 5 presents the approach for evaluating alternative design options with the adopted models and algorithmic solutions for the approach. Section 6 presents the tools that support the approach. Section 7 provides the discussion. Section 8 describes the related work and finally we conclude the paper in section 9.

2. BACKGROUND AND CONTEXT

In this section, we describe the background for understanding and supporting the approach that we present in this paper. In section 2.1 we present the common reference architecture for DDS, followed by a discussion in section 2.2 on the proposed DDS UML profile.

2.1 Reference Architecture for DDS

Based on DDS specification [26], we could derive the reference architecture for DDS based systems, as shown in Figure 1. A DDS system consisting of several DDS applications is called a *Domain*. A typical DDS based system is deployed on a number of *Application Nodes*. Each *Application Node* includes one or more *Domain Participants*, which are applications that together form the system execution. Each *Domain Participant* may include one *Publisher* that represents the objects responsible for data production and updates. A publisher includes one or more *Data Writers* that publish data of different data types. *Domain Participant* may also include one *Subscriber* that is responsible of receiving published data and making it available to the participant. A subscriber includes one or more *Data Readers* to access published data in a type-safe manner. Interaction between data reader and data writers is established via *Topics*. A topic defines a unique name, data type and a set of *Quality Services* to the published/subscribed data [26]. Note that Domain is a logical concept and a Domain Participant may participate to more than one domain at the same time.

Figure 1. Reference Architecture for DDS Based Systems

The DDS specification defines two layers: (1) A lower level layer, which provides efficient delivery of the shared information to the related recipients. This layer is named Data Centric Publish Subscribe (DCPS) in the specification and it is mandatory for the DDS

implementations. (2) A higher layer that enables simple integration of the services defined in DCPS layer into the application layer. This layer is named *Data Local Reconstruction Layer (DLRL)* in the specification and it is optional to be provided by the DDS implementations.

DDS provides the ability to specify various parameters like the rate of publication, rate of subscription, how long the data is valid, and many others. These Quality of Service (QoS) parameters allow system designers to construct distributed applications based on the requirements for, and availability of, each specific piece of data. Selected QoS parameters affect the performance of the overall system drastically, and therefore finding the feasible values for the QoS parameters for a system is important for successful development of the target system.

2.2 DDS UML Profile

To support the analysis and design of object-oriented systems using DDS technology, the OMG has specified the UML Profile for Data Distribution Specification [29]. The profile enables definition of all DDS artifacts defined in the reference architecture given in Figure 1. This profile also enables the definition of DDS data types which topics will be built on. The profile separates DDS artifacts in three packages including Data Centric Publish-Subscribe (DCPS), Data Local Reconstruction Layer (DLRL), and DDS Common. The DCPS defines the mandatory part of the DDS specification used to provide the functionality required for an application to publish and subscribe to the values of data objects. The DLRL is the optional portion of the DDS specification used to provide the functionality required for an application for direct access to data exchanged at the DCPS layer. The DDS Common package defines the distributed data communications specification that allows Quality of Service policies to be specified for data timeliness and reliability. It is independent of implementation languages. The dependencies between the packages are shown in Figure 2. The figure indicates that the DCPS and DLRL packages depend on DDS Common. Several tools that implement the draft specification of the above UML Profile for Data Distribution Specification are already available and ready to be used (e.g. Enterprise Architect [30]).

Figure 2. OMG UML Profile for Data Distribution Specification top-level packages

3. CASE STUDY – SMART CITY TRAFFIC SYSTEM

In this section, we define a case study that will be used to illustrate the problem statement and the approach in further sections. The case study that we consider is within the context of smart city engineering [31]. For the near future, it is expected that the gross of the world population will live in urban cities. This will have a huge impact on future personal lives and mobility. A smart city uses information and communication technology (ICT) to enhance the quality and performance of urban services, to reduce costs and resource consumption, and to engage more effectively and actively with its citizens [31], [32]. Sectors that have been developing smart city technology include government services, transport and traffic management, water and waste, health care, and energy. Smart city applications are developed with the goal to improve the management of urban flows and allowing for real time responses to challenges. One of the important applications in smart city engineering includes the development of smart traffic system (STS). Traffic is already a large problem in many cities and this problem will be even bigger in the future. Many people spend a considerable amount of time in traffic, which leads to unnecessary waste of human resource, time and increase of CO₂ emissions. STS provides

different capabilities such as traffic light management, congestion detection, traffic regulation, shared parking platform, etc. For example, shared parking platform optimizes the search for finding a suitable parking slot by guiding the drivers to the available and nearest parking spots in real-time.

The high-level reference architecture of STS is depicted in Figure 3. STS consist primarily of sensors and vehicles. Sensors are the devices that monitor the environment and provide the corresponding data. Vehicles use the sensor data and publish their position and other relevant information to the STS. Within the case study we distinguish between the following sensor types: *Traffic Light*, *Incident Detector*, *Congestion Detector*, *Speed Camera*, *Parking Detection Sensor*, *Bicycle Station*, *Parking Lot*, and *Weather Sensor*. Vehicles can be of the following types: *Car*, *Truck*, *Ambulance*, *Taxi*, *Bicycle*, and *Bus*. The sensors and control units are thin clients which do not contain any business logic. In this case, all the STS elements can communicate with the STS.

Figure 3. High Level Reference Architecture of the Smart City case study

STS is in essence a data-intensive system with stringent demands for QoS parameters. As stated before, the OMG's DDS Middleware explicitly considers QoS properties and as such is very suitable to realize the STS system. In order to implement STS using DDS we need to map the application domain (smart city) concepts to the DDS concepts, that is, domain, the domain participants, the publishers, the subscribers, and the topics in the STS case study. The DDS concept *domain* is here the *Smart City Traffic Domain*. Domain participants might be grouped as *vehicles*, *sensors* and *managers*. *Vehicles* and *sensors* are the virtual entities of the corresponding physical entities that we have described above. *Managers* define the domain participants that include the communication and business logic necessary for executing the required services. As stated before each domain participant can have zero or one publisher and zero or one subscriber. The subscribers and publishers for each domain participant are given in Table 1. For example, the entity *Car* has a corresponding domain participant *dpCar*, which as a publisher *dpCarPub* and a subscriber *dpCarSub*. In a similar sense the subscribers for each domain participant are defined. Finally, we have defined eight different topics that are shown in Table 2 which represents an example scenario using these topics. In the table we

can, for example, see that publisher *SpeedCameraPub* publishes data in the topic *Ticket Info Topic* with publish frequency rate 5Hz. The two subscribers are *TicketManSub* and *VehicleManSub* that read the published data. Table 3 shows an example scenario for STS including the defined number of instances per domain participant.

Table 1. Corresponding DDS Names for Application Domain Participants for STS

Application Domain Name	DDS Name	Publisher	Subscriber
Ambulance	dpAmbulance	dpAmbulancePub	dpAmbulanceSub
Bicycle	dpBicycle	dpBicyclePub	dpBicycleSub
Bus	dpBus	dpBusPub	dpBusSub
Car	dpCar	dpCarPub	dpCarSub
Taxi	dpTaxi	dpTaxiPub	dpTaxiSub
Truck	dpTruck	dpTruckPub	dpTruckSub
Bicycle Station	dpBicycleStation	dpBicycleStationPub	-
Congestion Sensor	dpCongestionSensor	dpCongestionSensorPub	-
Incident Sensor	dpIncidentSensor	dpIncidentSensorPub	-
Parking Lot	dpParkingLot	dpParkingLotPub	-
Speed Camera	dpSpeedCamera	dpSpeedCameraPub	-
Traffic Light	dpTrafficLight	dpTrafficLightPub	-
Weather Sensor	dpWeatherSensor	dpWeatherSensorPub	-
Incident Manager	dpIncidentManager	dpIncidentManagerPub	dpIncidentManagerSub
Logger Manager	dpLoggerManager	-	dpLoggerManagerSub
Parking Manager	dpParkingManager	dpParkingManagerPub	dpParkingManagerSub
Ticket Manager	dpTicketManager	dpTicketManagerPub	dpTicketManagerSub
Traffic Manager	dpTrafficManager	dpTrafficManagerPub	dpTrafficManagerSub
Vehicle Manager	dpVehicleManager	dpVehicleManagerPub	dpVehicleManagerSub
Whether Manager	dpWhetherManager	dpWhetherManagerPub	dpWhetherManagerSub

Table 2. Topics of Sample Scenario for Smart Parking System (STS)

Topic Name	Publisher	Publish Rate [Hz]	Subscriber
Vehicle Info Topic	dpCarPub	5	VehicleManSub
	dpBusPub	5	
	dpTruckPub	5	
	dpAmbulancePub	5	
	dpTaxiPub	5	
	dpBicyclePub	5	
Global Info Topic	VehicleManPub	20	dpCarSub dpBusSub dpTruckSub dpAmbulanceSub dpTaxiSub dpBicycleSub
Traffic Info Topic	TrafficLightPub	10	TrafficManSub
	CongestionSensorPub	10	VehicleManSub
Ticket Info Topic	SpeedCameraPub	10	TicketManSub VehicleManSub
Whether Info Topic	WhetherSensorPub	10	WhetherManSub TrafficManSub VehicleManSub
Parking Info Topic	BicycleStationPub	10	ParkingManSub
	ParkingLotPub	10	VehicleManSub
Incident Info Topic	IncidentSensorPub	10	IncidentManSub VehicleManSub
Logger Topic	TrafficManPub	20	LoggerManSub
	TicketManPub	20	
	WhetherManPub	20	
	ParkingManPub	20	
	IncidentManPub	20	

Table 3. Example scenario for STS with defined number of instances per domain participant

Domain Participant Name	Number of instances
Ambulance	17
Bicycle	184
Bus	46
Car	1435
Taxi	124
Truck	28
Bicycle Station	23
Congestion Sensor	62
Incident Sensor	29
Parking Lot	33
Speed Camera	48
Traffic Light	125
Weather Sensor	16
Incident Manager	15
Logger Manager	15
Parking Manager	15
Ticket Manager	15
Traffic Manager	15
Vehicle Manager	15
Whether Manager	15

4. Problem Statement

An important step of designing a DDS-based application is to define the deployment model of the system. The deployment model defines the allocation of domain participant instances (e.g. scenario of Table 3) to the available physical resources, and largely influences the performance of the overall system. In principle many different deployment alternatives can be

defined. For example, a deployment alternative of the STS can be defined with three nodes in which all vehicle instances are deployed on the first node, sensor instances are on the second node and manager instances are deployed on the third node as given in Figure 4. Actually, this alternative follows the conceptual separation of concerns in which a separate node is logically defined almost for each participant type. Further, the communication overhead among the same participant types such as the communication between Vehicle Manager and Traffic Manager are minimized because of being deployed on the same node. Although this alternative is easy to understand because of the logical separation of participants, it does not always have good time performance because separately deployed participants such as parking sensors and vehicles need to interact very frequently with each other.

Figure 4. Deployment by grouping domain participants

A second deployment alternative example can be defined as we evenly distribute and deploy the participants over the ten nodes as given in Figure 5.

Figure 5. Deployment by distributing domain participants over nodes.

This is a trivial deployment alternative which might not be optimal in case nodes are not homogenous and have different memory and processing power. We can derive many more different deployment alternatives that may differ with respect to the number of deployment nodes and the mapping of participants to the nodes. Apparently, the number of deployment alternatives is very large and each deployment alternative will perform different with respect to different quality considerations such as logical separation for understandability, optimizing communication overhead, enhancing utilization of physical resources, etc. Obviously, a more

systematic and formal approach is required to guide the search for the feasible deployment alternatives. The OMG DDS specification does not provide an explicit approach to guide the distribution and allocation of the participants to optimize the deployment model with respect to performance in the design phase. Moreover, currently there is no adequate approach and tool support yet to enable the selection of deployment alternatives in the literature. In the following sections, we will provide an approach and tool framework for designing the DDS-based application and deriving feasible deployment alternatives.

5. APPROACH FOR GENERATING DDS DEPLOYMENT CONFIGURATION ALTERNATIVES

In this section, we provide a systematic process for defining and evaluating feasible deployment alternatives of a DDS-based distributed system. The presented approach will be used in the design phase of the DDS-based system where the development of the system is not started yet, and the system code is not available. The approach is represented as an activity diagram as shown in Figure 6. The approach consists of the two basic phases “Architecture Design” and “Feasible Deployment Generation”.

Figure 6. Activity Flow of Alternative Design Evaluation and Deriving Feasible Deployment

Typically, the architecture design phase follows the requirements analysis process. We assume that the requirements analysis phase is performed using the approaches as defined in the literature (e.g. see Rational Unified Process [33]) and provides the input for the DDS-based system architecture. Therefore, this first step is shown in grey in the figure.

The architecture of the DDS application is designed using the DDS UML Profile that has been defined in section 2.2. This includes the definition of the DDS Types, the DDS Topics, the Domain Participants and the Publish/Subscribe Relations. The DDS application will be deployed on the target environment, which consists of physical resources on which the DDS

domain participants will execute. The design of the physical resources is defined in parallel to the DDS application design.

After the architectural system design phase is completed, the feasible deployment model generation phase starts with the definition of the execution configuration. The execution configuration defines the number of each DDS domain participant and update rate for each publication by using the artifacts defined in architecture design phase. From an abstract point of view, the feasible deployment models of a system with several sub-components can be derived by using task assignment algorithms defined in the literature. For utilizing the task assignment algorithms, the required parameters will be extracted from the design. After the necessary input parameters are generated, the feasible deployment models are defined and the deployment models are generated. Subsequently, the feasibility of the generated deployment models is evaluated in the following step. If the generated deployment models are not satisfactory, an iteration step will be required to analyze the system design and refine it according to the provided feedback by the corresponding tool. Here a satisfactory alternative defines a deployment alternative that meets the expected improvement rate of the costs (e.g. communication and execution costs) for the deployment model. Finding feasible deployment models may require several iterations of the process steps. The initial deployment model is realized and verified in development and integration/test activities, and the results are fed back to the designer until a satisfactory alternative is derived.

In the following subsections we will explain the concrete activities that we have defined to realize our approach. Each section also defines the metamodels that are used for modeling the related artifacts of the corresponding step.

5.1 Define DDS Application

OMG's UML Profile for Data Distribution Specification already defines necessary metamodel for defining a DDS application, so we did not define a new metamodel for DDS Application definition.

The approach defined in this paper extends and realizes the OMG UML Profile for Data Distribution Specification [29]. The core classes of this profile are given in Figure 7. All other classes are derived from these basic artifacts. Hereby, *Entity* defines an abstract class, specialized by all entities defined in the metamodel. *Specification* defines a container of properties, and a constraint on the range of values represented by a typed element. Finally, *TypedEntity* defines an entity that has a type that serves as a constraint on the range of values the entity can represent.

Figure 7. Metamodel for DDS UML Profile/Common/Core Package

Our modeling tool realizes necessary parts of UML Profile for Data Distribution Specification to define the DDS types, the DDS topics, the Domain Participants, and the Publish/Subscribe Relations. For example, the relationship among *Domain*, *Domain Participant*, *Publisher*, *Subscriber*, *Data Reader*, and *Data Writer* artifacts are shown in Figure 8. The model implies that a DDS application may consist of one or more *Domains*, a *Domain Participant* can be member of one or more *Domains*, a *Domain Participant* may contain zero or one *Publisher/Subscriber*, and so on. The attributes of metamodel classes are not shown for the sake of simplicity and can be inspected from the specification [29]. All of the metamodels that

we have adopted from OMG's UML Profile for Data Distribution Specification are shown in Appendix A.

Figure 8. Metamodel for DDS UML Profile/DCPS/DCPS Package

5.2 Design Physical Resources

Parallel to the activity *Define DDS Application*, the activity *Design Physical Resources* defines the available nodes together with their processing power and memory capacity, as well as the network connections among the nodes. For example, one may decide to adopt 25 nodes on which the participants need to be deployed. Further it could be decided that each node has a memory capacity of 12280 MB and contains two processing units with four cores at the frequency of 2.3 MHz. Equally, the nodes could also have different memory capacity and computation power.

The physical resource metamodel has not been defined in the UML Profile for Data Distribution Specification. As such, we have developed the metamodel in Figure 9 to support the process in Figure 6. The Physical Resource Metamodel given in Figure 9 can be used to represent the artifacts for modeling the available physical resources.

Figure 9. Physical Resource Metamodel

PhysicalResourceModel is the root class of the metamodel that defines a physical resource model. There can be one or more *Nodes* in a physical resource model, which represents computation resources. Each node has a *name* attribute that identifies the node. The *powerFactor* attribute defines the computation power of the node relative to other nodes. A node can have one or more processors, one or more custom node properties, and memory capacity. *Processor* defines properties of a processing unit using the attributes *name*, *frequency* and *coreCount*. The attribute *name* is the symbolic name of the processor like “Intel Core I7”. The attribute *coreCount* defines the number of cores that the processor has. The attribute *frequency* defines the frequency of the processor in Mhz. *MemoryCapacity* has a *value* attribute that represents the memory capacity of the node in terms of megabytes. *CustomNodeProperty* can be used to define additional properties for the node. The properties are defined as name-value pairs. For example, one may decide to include a specific property *diskCapacity* with value 340 Gb.

There can be one or more networks in a physical resource model. The *Network* class is the abstract base class for *LocalAreaNetwork (LAN)* and *WideAreaNetwork (WAN)* classes. The *name* attribute of the *Network* class is the symbolic name of the network. *WideAreaNetwork* class has *speedFactor* attribute that defines the speed of the network in comparison with a LAN. *LANConnection* represents the connection of a node to a LAN. *Router* represents routers for connecting networks with each other. The *name* attribute of the *Router* class is the symbolic name of the router. *LANRouterConnection* class represents connection of a LAN to a router while the *RouterNetworkConnection* class represents connection of a router to a network.

5.3 Design Execution Configuration

The *Execution Configuration Metamodel* is used to define the artifacts to model the execution configuration shown in Figure 10. *ExecutionConfiguration* class defines an execution configuration which contains elements of *Metadata* and *DomainParticipantInstance*. *Metadata* defines *name*, *creation date*, *creator*, and *version* of the execution configuration. *DomainParticipantInstance* represents an instance of a Domain Participant that is defined in the *DDS Application Definition Metamodel*.

Each Domain Participant instance can have a different execution cost for different nodes. For this, *DomainParticipantInstance* contains a list of *ExecutionCost* that define estimated execution cost for each node which the Domain Participant instance can execute. Note that the execution cost is dependent on the selected execution configuration. For example, the execution cost of a *Mobile Client Subscriber* model changes according to existing *Parking Detection Sensors* in the execution configuration. The execution cost is a scaled value that shows the execution cost of a Domain Participant Instance in comparison with other Domain Participant Instances in the execution configuration. For example, the execution cost for each *Parking Detection Sensor* module instance is defined using scaled value and defined as 7 over 20 for one node, 14 over 20 for another node, etc. The execution costs of modules are influenced by the processor’s *powerFactor* and *memoryCapacity* attributes. In a similar sense, the communication costs among modules are influenced by the networks *speedFactor* attribute. Since the execution and communication costs of module instances can only be exactly measured after the system is developed, during design time their values can only be estimated [15]. This estimation can be conducted by using, for example, design phase complexity calculation methods such as proposed by [19] or prototyping.

The attribute *requiredMemory* of *DomainParticipantInstance* represents the estimated memory amount that the module instance will require during execution. Similar to the execution cost, this parameter can be estimated in the design phase. The attribute *instanceCount* defines the number of Domain Participant Instances in the execution configuration. This attribute is added because there may be multiple instances of the same Domain Participant in an execution configuration. For example, in a large Smart Parking System scenario, there can be hundreds of *Parking Detection Sensors* and it is not feasible to add one module instance for each of them to the execution configuration separately.

The relation *relatedDomainParticipant* associates a *DomainParticipantInstance* with a *DomainParticipant* that is defined in the activity Define DDS Application. *DomainParticipantInstance* can have zero or more *Publications* that represent the update rate and the related element from DDS Topic definition. Each publication is associated with an *TopicDescription* defined in “Define DDS Application” step.

The *updateRate* attribute shows how many times a Domain Participant instance will update a *Topic* in a second. For example, we could decide to have 2000 *Parking Detection Sensor* module instances where each of them publishes a *Sensor* object with update rate of 2 times per minute.

Figure 10. Execution Configuration Metamodel

5.4 Generate Input Parameters for Allocation Algorithm

Once the parameters for the physical resources and execution configurations have been defined we can start the search for the feasible deployment alternatives. In principle, this can be carried out in different ways in which multiple different approaches and algorithms can be identified. The allocation could be, for example, based on the following heuristics:

- (1) minimizing the number of the nodes to which the tasks are allocated
- (2) uniform distribution of tasks over the nodes
- (3) just random allocation of tasks over the nodes
- (4) minimizing the overall communication costs

The presented approach is generic and does not hardwire a particular heuristic approach. If needed, in addition to the above heuristics we could also identify other heuristics. In the next section, we will discuss each of these approaches in the implementation of the tool and the overall evaluation.

Besides of heuristic approaches we could also adopt a more formal and systematic algorithm for the deployment process. In this paper, we will adopt the so-called Multi-Processor Task Assignment (MPTA) problem [24][16]. For this problem, the following parameters can be defined:

- T , set of m tasks = $\{t_1, t_2, \dots, t_m\}$
- P , set of n processors $\{p_1, p_2, \dots, p_n\}$
- M_p , memory capacity of processor p
- m_i , amount of memory needed for task i
- X_{iq} , cost of executing t_i task on p_q processor.
- E , set of communication between tasks, whereby each communicating task combination (i, j) has a communication cost c_{ij} if tasks t_i and t_j are assigned to

different processors. Communication cost is negligible if two tasks are assigned to same processor.

The objective in our problem is to minimize the sum of total execution cost and total communication cost (among simulation module instances) while not exceeding the memory capacity of each node. Based on the above definitions we can formulate our objective as follows [16][24]:

<p style="text-align: center;"><i>Assign tasks to processors to minimize the sum</i></p> $\sum_{i=1}^m \sum_{p=1}^n a_{ip} x_{ip} + \sum_{(i,j) \in E} \sum_{p=1}^n a_{ip} (1 - a_{jp}) c_{ij}$ <p style="text-align: center;"><i>Subject to</i></p> $\sum_{p=1}^n a_{ip} = 1, i \in T$ $\sum_i m_i a_{ip} \leq M_p$ $a_{ip} = \{0,1\}, p \in P, i \in T$ <p style="text-align: center;"><small>($a_{ip} = 1$, if task i is assigned to processor p, 0 otherwise)</small></p>
--

In fact, the required parameters of the MPTA problem can be extracted from the simulation design that has been defined in the previous activities. In Table 4 we explain for each parameter how it is extracted from the design.

Table 4. Extracting MPTA parameters from the design

MPTA Parameter	Extraction from design
T	List of module instances defined in Simulation Execution Configuration Development activity.
P	Each <i>node</i> defined in Physical Resource Design activity.
M_p	<i>memoryCapacity</i> attribute of <i>node</i> defined in Physical Resource Design activity.
m_i	<i>requiredMemory</i> attribute of <i>ModuleInstance</i> defined in Simulation Execution Configuration Development activity.
X_{iq}	<i>nodeExecutionCostTable</i> attribute of <i>ModuleInstance</i> defined in Simulation Execution Configuration Development activity.
C_{ij}	Calculated by using: <ul style="list-style-type: none"> - <i>Publications</i> defined in Simulation Execution Configuration Design activity, - <i>Subscriptions</i> defined in Publish/Subscribe Relations of Simulation Modules Design activity, - <i>Object model elements</i> defined in Simulation Data Exchange Model Design activity

5.5 Find Feasible Deployment Configuration

The activity *Find Feasible Deployment* takes as input the parameter values of the previous activity and executes an algorithm that computes a feasible deployment alternative, if one is available. Different algorithms in the literature can be used to solve the MPTA problem. Please note that we do not focus on a particular algorithm but recommend using a practical one for the corresponding case. In our case, we could for example use the MPTA algorithm as defined by Mehrabi et al. [16] because it adopts the parameters of execution cost, communication cost and memory requirements. If a feasible deployment is found, the output of this activity is a table that represents the mapping of tasks (module instances) to processors (nodes). If the algorithm was not successful in finding a feasible solution the process returns to the activity *Develop Simulation Execution Configuration*. This can be repeated several times until a feasible deployment is found. If it appears that a feasible deployment cannot be found by changing just the simulation execution configuration, then the designer can decide to return to the beginning of step 3 to refine/update the design.

5.6 Generate Deployment Configuration

The *Deployment Metamodel* is used to describe the deployment model in the “Generate Deployment Model(s)” activity shown in Figure 11. The deployment metamodel contains *Members* and *Nodes*. Each *Member* is deployed on one of the *Nodes* defined in *Physical Resource Model*. One or more *Domain Participant Instances* can be deployed on a *Member*.

Figure 11. Deployment Configuration Metamodel

6. TOOLS AND APPLYING THE APPROACH TO THE CASE STUDY

In this section, we present the tool *DEPLOY-DDS* that provides an integrated development environment for supporting the activities of the approach described in the previous section. *DEPLOY-DDS* is built on the Eclipse platform and is implemented as a set of plug-ins. The developed plug-ins are built on other Eclipse frameworks including Eclipse Modeling Framework (EMF) [4], and Graphical Modeling Framework (GMF) [25]. EMF is a modeling framework and code generation facility that we use to develop the metamodels. GMF is a generative component and runtime infrastructure that we use for developing graphical editors for the developed metamodels. Further, we use *Emfatic* [7], which provides a text editor and a language for editing EMF models. In addition, we use *EuGENia* GMF tool [11] that provides mechanisms for abstracting away the complexity of GMF and for easier development of GMF editors. *EuGENia* tool is a part of *Epsilon* project [13].

In the following subsections, we describe the top-level tool architecture in section 6.1. In section 6.2 we show the application of *DEPLOY-DDS* for designing the DDS Application, Physical Resources, Execution Configuration for the case study.

6.1 Tool Architecture

The common perspective of *DEPLOY-DDS* is given in Figure 12. The left pane includes the Model Explorer View that shows the available models and their elements. The Editing pane in the middle provides the main drawing area for the DDS based application design. The Properties Editor View at the bottom provides an editing area for the attributes of the design model elements that are selected from the Editing Pane or the Model Explorer.

Figure 12. General Perspective of DEPLOY-DDS tool

DEPLOY-DDS supports different activities in the approach; the dependencies between these activities are shown in Figure 13. The activities result in artifacts which are denoted using the stereotype <<Artifact>>. The circles with numbers denote the control flow among the activities.

The *DDS Type Repository Definition* results in the *DDS Type Repository*, which is provided as an input to the *DDS Topics & Participants Definition* activity, and *Execution Configuration Definition* activity. The *DDS Topics & Participants Definition* activity is used to produce the *DDS Topics*, *Domain Participants*, and *Pub/Sub Definitions* which is also an input to the *Execution Configuration Definition* activity. The *Physical Resources Design* activity is used to define the *Physical Resource Model*, which is an input to *Execution Configuration Definition* activity and the *Deployment Model Generation* activity. The *Execution Configuration Definition* activity is used to define the *Execution Configuration*, which is provided as an input to the *Deployment Model Generation* activity that on its turn generates the *Deployment Model*. In the following subsections, we describe each activity in more detail using the Smart Parking System (STS) case study defined in Section 3.

Figure 13. Dependency Graph of Activities

6.2 Using *DEPLOY-DDS* to design DDS Application Models for the Case Study

As stated before, using the tool the activities *DDS Application Design*, *Type Repository Definition*, *Physical Resources Design* and *Execution Configuration Design* define the corresponding modeling tools.

Figure 14 shows a part of the DDS Type Repository of the case that has been developed using the *DDS Type Repository Definition* activity. As stated before publishers and subscribers communicate via topics. Hereby, publishers write data fields in the topic and subscribers read data fields in the topic. Type Definitions of the topics are given in Figure 14. For example, in this diagram we defined a topic *struct VehicleInfo*. In this topic we have four data fields. The *<int> vehicleID* field shows the unique ID of the related vehicle. The *<int> speed* field shows the speed of the vehicle. Finally, *<double> latitude* and *<double> longitude* fields show the geographic position of the vehicle.

Figure 14. Type Definition Model of the case study

The Application Definition diagram of the case study is given in Figure 15.

Figure 15. Application Definition Model of the case study

Figure 15 shows the application definition model of the case study. The domain participants in this figure are all in the “*Smart City Traffic*” (STS) domain. In this diagram, we can classify domain participants mainly in three categories: Vehicles, Sensors and Managers. Sensors just have data publishers to publish related sensor information. This information is read by related managers and vehicles via the defined topics. Vehicles have both publishers and subscribers. They publish their id, speed and position information basically. This information is read by

managers. Similar to vehicles, managers have publishers and subscribers except *Logger Manager* which has nothing to publish in STS domain and just a subscriber. Managers combine this information with information coming from sensors. Managers might communicate with other software modules such as database and cloud modules (for simplicity this part was excluded in the model). Resulting information combined in managers are published into vehicles again and drivers might have broad information about the details of the city traffic such as accident information and congestion information so that they can use less dense roads or they can arrive proper parking places with less trip.

Figure 16. Physical Resource Model for Case Study with Ten Nodes

Figure 16 reveals the Physical Resource Model Diagram of the case study. In this case, we have 10 nodes (computers) with different number of processors and different memory capacities. This heterogeneity makes obtaining feasible solution more difficult.

Figure 17 shows the execution configuration model for the Vehicle participants of the case study. Hereby, as an example Vehicle publishers publish data at 5 Hz, with different execution costs for different nodes. For this paper, we assume that the proper execution costs are provided. These could be typically obtained experimentally or based on expert knowledge. The more precise the values of the execution costs the more effective the tool will be to derive the feasible deployment alternatives.

Figure 17. Partial view of the Execution Configuration Diagram for the Case Study

7. EVALUATION

In the previous sub-sections, we have described the development of the physical configuration model, the type definition model, the application definition model and the execution configuration model. Given these models we can now generate the possible deployment alternatives. The corresponding snapshot of the tool is shown in Figure 18. As it can be seen in the figure the execution configuration, and the physical resource model can be provided as an input to the tool. The field *Container*, defines the folder in which the results are stored. In principle, the deployment generation can be realized using multiple different alternative algorithms. The user can select one of the implemented deployment model generators.

In *DeployDDS* tool, we have selected five different deployment model generators (DMG) to obtain the deployment models including *DMG_TopicBasedAllocation*, *DMG_GeneticAlgorithm*, *DMG_SequentialAllocation*, and *DMG_MinimumNodeAllocation*. Each of these algorithms has been implemented in the tool and provides a solution for the MPTA problem as discussed in section 5.4. In the following, we shortly describe the algorithms that we have implemented:

Figure 18. Algorithms used to find Feasible Deployment Alternatives in DeployDDS tool

DMG_TopicBasedAllocation aims to find feasible deployment models with minimum communication cost. This logic is implemented by a Greedy Algorithm which allocates the publishers and subscribers of the same topic into the same node. If the node does not have adequate memory for the publishers and the subscribers, only appropriate number of publishers and subscribers will be allocated to that node. The number of nodes that cannot be allocated to the same node because of the lack of memory will be allocated to the next nodes. As stated above, the publishers and the subscribers that cannot be allocated to the same node will cause communication cost. So, if there is enough memory to allocate all communicating publishers and the subscribers into the same node, this DMG will result in zero communication cost.

DMG_GeneticAlgorithms uses a genetic algorithm based solver to find feasible deployment models. The details of the genetic algorithm are given in section 6.3 and [29].

DMG_SequentialAllocation, allocate domain participant instances into the available nodes. It starts with the first domain participant, allocate sufficient number of domain participants into the first node. It will allocate sufficient number of participants to the first available nodes and then switches to the second node. Note that, if the memory available in the first node is

sufficient to allocate the all participants, then this DMG will result the same deployment model with DMG_MinimumNodeAllocations.

DMG_MinimumNodeAllocation aims to find feasible deployment models using the minimum number of nodes. If possible, this DMG allocates all tasks into the same node which will result in zero communication cost. In order to allocate all participants into the same node, this DMG starts from the node that has maximum memmory available. If the memory required to allocate all tasks into the same node is not available in a single node, then more nodes will be allocated. The resulting deployment model will be using the minimum number of nodes. At the end, many nodes might become unused.

By selecting, one of these generators the feasible deployment alternative can be automatically generated using the selected deployment generator. If necessary, the user of the tool can implement another algorithm and deploy it in the tool. In principle, each newly defined algorithm will follow the steps of the common pseudo-code as shown in Figure 19. As shown in line 1, the algorithm *GENERATE_FEASIBLE_DEPLOYMENT* takes two input parameters: a physical resource model and a simulation execution configuration as defined, for example, in Figure 16 and Figure 17, respectively. Line 2 extracts processors from the physical resource model by calling *EXTRACT_PROCESSORS* in which a processor is created for each node in the physical resource model. In Line 3, tasks are extracted from the simulation execution configuration by calling *EXTRACT_TASKS* in which a task is created for each module instance and execution cost among tasks is calculated. In Line 4, the actual MPTA algorithm is executed by calling *EXECUTE_MPTA*. The result of this is stored in *assignment_table* that includes the assignment of tasks to the processors. Likewise, *assignment_table* defines an abstract specification of the feasible deployment alternative. In Line 5, the deployment is actually generated by calling *CREATE_DEPLOYMENT_MODEL* with the parameter *assignment_table*.

1.	<i>GENERATE_FEASIBLE_DEPLOYMENT</i> (<i>phy_resources</i> , <i>exec_config</i>)
2.	<i>processors</i> $\leftarrow$ <i>EXTRACT_PROCESSORS</i> (<i>phy_resources</i>)
3.	<i>tasks</i> $\leftarrow$ <i>EXTRACT_TASKS</i> (<i>exec_config</i>)
4.	<i>assignment_table</i> $\leftarrow$ <i>EXECUTE_MPTA</i> (<i>tasks</i> , <i>processors</i>)
5.	<i>CREATE_DEPLOYMENT_MODEL</i> (<i>assignment_table</i>)

Figure 19. Pseudo-code for generating feasible deployment alternative

Figure 20 shows the generated deployment alternatives for the case study using the *DMG_TopicBasedAllocation* [16]. The generation algorithm is implemented in Java and executed on a quad-core Intel I-5 2.70 GHz 64-Bit computer with 4 GB of RAM. As it can be observed the resulting deployment model includes 10 nodes as given before in the physical resource definition model in Figure 16. Further, the execution configuration model as partially defined in Figure 17 has been deployed to the physical nodes to optimize the values for the metrics execution cost, communication cost and memory requirements. A close analysis of the generated alternative of Figure 20 shows that the total memory requirements of domain participant instances that are deployed on each node do not exceed the memory capacity of the corresponding nodes. Further, based on the adopted genetic algorithm, it appears that domain participant instances that interact frequently and which have high communication costs, are as much as possible co-located on the same node. The domain participant instances that are remaining and which would exceed the memory capacity of Node-1 are deployed to other nodes in a similar manner. Overall, the feasibility of the generated deployment alternative is based on the MPTA algorithm that we have used and which has been validated in earlier studies [16].

Figure 20. Generated Feasible Deployment Alternative including 2275 Tasks with DMG_TopicBasedAllocation

The generated deployment diagram can soon become too large to view in a single diagram. For this we can also show the results in a table as shown in Table 5. The results for the selection of the other deployment generator algorithms are shown in Table 10 (*DMG_GeneticAlgorithm*), Table 12 (*DMG_SequentialAllocation*), and Table 11 (*DMG_MinimumNodeAllocation*) of Appendix B.

Table 5. Deployment Results for DMG_TopicBasedAllocation

Instance Name	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	TOTAL
dpIAmbulance									15	2	17
dpIBicycle										184	184
dpIBus										46	46
dpICar					320	67	426	426		196	1435
dpITaxi						124					124
dpITruck						28					28
dpIBicycleStation									23		23
dpICongestionSensor									62		62
dpIIncidentSensor									29		29
dpIParkingLot									33		33
dpISpeedCamera									48		48
dpITrafficLight									47	78	125
dpIWeatherSensor									15	1	16
dpIIncidentMan									15		15
dpILoggerMan									15		15
dpITicketMan									15		15
dpITrafficMan									15		15
dpIParkingMan									15		15
dpIVehicleMan									15		15
dpIWeatherMan									15		15
TOTAL	0	0	0	0	320	219	426	426	377	507	2275

Each of these deployment generators will perform differently. To validate each algorithm, we adopt the *Communication Cost* and *Execution Cost* metrics. *Communication Cost* defines the overall communication costs of the required communication tasks in the generated deployment alternative. The execution cost metric defines the overall cost of the required

tasks on the required number of processors. We have calculated the communication costs and execution costs for the selected deployment generators applied to the the case study. The results are shown in Table 6.

Table 6. The communication and execution costs values for the selected Deployment Generators

Deployment Generator	Communication Cost (Mbytes/s)	Execution Cost (Mbytes/s)
DMG_TopicBasedAllocation	4.05	27773
DMG_GeneticAlgorithm	5.00	31381
DMG_MinimumNodeAllocation	4.02	27730
DMG_SequentialAllocation	4.76	31763

As we can observe in Table 6 the overall communication cost value is acquired by the generator *DMG_TopicBasedAllocation*, while *DMG_GeneticAlgorithm* performs worst. For the execution cost metric the deployment generator *DMG_MinimumNodeAllocation* performs the best, while *DMG_GeneticAlgorithm* has the lowest performance. These values are also shown by the tool and as such provide useful insight for deciding on the proper deployment allocation.

Each of the selected algorithms provide feasible deployment alternatives and can in principle directly used to implement the system. In order to further analyze the validity of the generated deployment models we use two approaches.

The first approach is intuitive and based on the visual inspection of the generated deployment model alternatives by an expert. Therefore, this approach relies on the experts experience to provide logical reasoning about the feasibility of the deployment alternative. In addition, the generation of the alternative is done automatically and not performed by the expert. An example reasoning of an expert could be based on the deployment alternative given in Figure 20. A close analysis of this generated deployment alternative shows that the total memory requirements of DDS based software system (i.e., STS) does not exceed the capacity of the corresponding nodes. Further, based on the adopted genetic algorithm, it appears that software module instances that interact frequently and which have high communication costs, are as much as possible co-located on the same node. Apparently, the publishers and subscribers in the system have frequent interactions in publish-subscribe communication via ParkingLot topic and in the deployment model (Figure 20). The adopted algorithm has co-located instances of these modules as much as possible to keep the communication cost minimum. The remaining instances, which would exceed the capacity of Node-1, are deployed to other nodes in a similar manner.

The second, more formal approach for evaluating the generated deployment alternative is to compare the generated alternative with another deployment alternative. As shown in Figure 21, the DeployDDS tool enables the comparison of two deployment models that were defined before, either generated and/or manually defined. To compare two models, the execution configuration and the physical resource model is provided. Once the *Compare* button is pressed the output is written to the corresponding result folders.

Deployment Model Evaluator
This wizard evaluates a deployment model by comparing it with another reference model.

Execution Configuration: Browse...

Physical Resource Model: Browse...

Original Deployment Model: Browse...

Optimized Deployment Model: Browse...

Results File Container: Browse...

Results File Name:

Compare Cancel

Figure 21. Deployment Model Evaluator of DeployDDS tool

The comparison process provided in the DeployDDS is generic and can be applied in a similar way for the alternatives generated with all the defined deployment generators. We show the evaluation of the generated deployment model with a manually generated deployment model that is based on a deployment model that is generated by an expert. We have manually defined the deployment model for the expert judgment deployment alternative in DeployDDS environment. The results of the expert allocation are shown in Table 7.

Table 7. Deployment Results for Expert Distribution

Instance Name	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	TOTAL
dpiAmbulance									17		17
dpiBicycle									184		184
dpiBus								16	30		46
dpiCar	17	17	123	123	230	231	338	322	17	17	1435
dpiTaxi	12	13	13	13	13	12	12	12	12	12	124
dpiTruck	3	2	2	3	3	3	3	3	3	3	28
dpiBicycleStation										23	23
dpiCongestionSensor										62	62
dpiIncidentSensor										29	29
dpiParkingLot										33	33
dpiSpeedCamera									6	42	48
dpiTrafficLight									125		125
dpiWeatherSensor									16		16
dpiIncidentMan										15	15
dpiLoggerMan										15	15
dpiTicketMan										15	15
dpiTrafficMan										15	15
dpiParkingMan										15	15
dpiVehicleMan										15	15
dpiWhetherMan										15	15
TOTAL	32	32	138	139	246	246	353	353	410	326	2275

The expert deployment allocates an equal number of domain participant instances to each node. The expert checks the available memory of the nodes and if the memory is not sufficient for the required number of tasks, he/she tries to allocate the remaining tasks to the other available nodes. The results for this expert deployment allocation are shown in Table 7. Figure 22 shows the results of the expert deployment for three nodes. The numbers in each cell defines the number of instances of the participants. For example, dpiBicycleStation (x23) means that 23 instances of BicycleStation is deployed into the node and dpiAmbulance means a single instance of Ambulance is deployed into the corresponding node. The communication and execution costs values for this expert deployment is given in Table 8.

Figure 22. Expert Deployment Model for first three nodes

Table 8. The communication and execution costs values for Expert Deployment

Deployment Generator	Communication Cost (Mbytes/s)	Execution Cost (Mbytes/s)
Expert Deployment	4.37	29483

If we compare the expert based deployment with that of the earlier defined DMG_TopicBasedAllocation in Figure 20 we can conclude that both the communication costs and execution costs metric values are slightly better for the DMG_TopicBasedAllocation. DMG_TopicBasedAllocation tries to define the deployment such that the participants and the subscribers of the same topic is located into the same node. This strategy minimizes the communication cost in the deployment model. For example, the participant BicycleStation publishes the data which is subscribed by ParkingManager. As such, these two participants are located into the same node. Similarly, ParkingLot and ParkingManager are located into the second node and ConcessionSensor and the TrafficMan participants are deployed into the third node.

Similar to the comparison with DMG_TopicBasedAllocation we can compare the expert deployment also with the results of the other deployment generators. The comparison results are shown in Table 9. Here we have set the communication cost and execution costs of the expert deployment to 100%. The other percentages define the percentage in relation to the expert's results. From this table, we can conclude that the defined deployment generators

perform in general better than the expert deployment. When execution costs are compared the DMG_MinimumNodeAllocation seems to perform the best. Based on these results a given deployment generator could be selected. Note that the results of the algorithms can be different for different execution configuration model and the physical resource model. The approach and the tool can be used to assist in selecting the most feasible deployment model. As stated before, if needed, new deployment generators can be easily defined to optimize the results even further.

Table 9. Comparison of Expert Deployment Models wrt Deployment Model Generators

Deployment Generator	Communication Cost (%)	Execution Cost (%)
DMG_TopicBasedAllocation	92.8	94.1
DMG_GeneticAlgorithm	114.4	106.4
DMG_MinimumNodeAllocation	92.1	94.1
DMG_SequentialAllocation	108.9	107.7
Expert Deployment Model	100.0	100.0

8. DISCUSSION

The Data Distribution Service (DDS) is now a popular and recognized data-centric publish-subscribe programming model and specification for distributed systems. It has been applied in many different application domains which has resulted in several lessons learned. One of the important issues is the support for modeling and design abstractions in DDS based systems. To OMG has provided the DDS UML Profile to support the analysis and design of a DDS-based distributed system. The focus of the paper has been mainly on deriving configuration alternatives. This is an important and relevant problem for many DDS-based systems which consists usually of multiple participant applications each of which has different responsibilities in the system. The potential configuration space is in general too large and not tractable for the human system engineer and a systematic approach with automated support is necessary. We have provided both a systematic approach with the related toolset that can be used for a broad range of DDS-based systems to derive feasible configuration alternatives to meet the functional and quality concerns given the available resources. The approach has also been illustrated for a relevant case study on smart city engineering which has been used to illustrate both the problem and the approach.

The approach adopted the UML profile to complement with the existing work. The UML profile appeared to be very useful in preparing and supporting the analysis and design of the DDS system for deriving design alternatives. It should be noted that the OMG's DDS UML Profile is a specification and no realization of it was present yet. As such, one of the supporting contributions in this paper is also the realization of this profile in the Eclipse Modeling Framework (EMF). To support the systematic approach for generating the design alternatives we had to enhance the profile further (e.g. physical resource modeling, and execution configuration model).

Based on the modeled system design and physical resources using the realized DDS's UML Profile, the feasible deployment alternatives could be algorithmically derived and automatically generated using the developed tools. In the toolset, we have implemented different algorithms for deriving feasible deployment alternatives. Yet the approach does not mandate the usage of a particular algorithm but provides the required input values for these algorithms. The focus of the paper is not the design of algorithms but the overall system engineering approach for selecting a feasible alternative in a large configuration space. The algorithms that we used were justified in the literature for solving the MPTA problem as discussed in section 5. The correctness of these algorithms has been discussed in the corresponding papers and based on this we can assume that a feasible solution is derived. In addition, depending on the state of the system different MPTA algorithm implementations may be used to optimize the metric values. For comparison of the algorithms, we refer to, for example, [24].

Both the approach and the tools assist the designer to derive a feasible deployment model. We do not maintain that the tool is a replacement for the human expert. In fact, the tool can be a complementary and supporting alternative for the human expert who can design, generate and evaluate the derived alternatives. After deriving the deployment alternative, if necessary, expert judgment can be further used to refine the deployment alternative.

One of the important benefits of the approach is also the early analysis of the system and the generation of the feasible deployment model at design time. Deferring the definition of the deployment to the development phase might in practice easily lead to non-feasible implementations which will require iterating the design and the related project lifecycle artifacts such as detailed design, implementation, test artifacts, documentation, etc.

The identified deployment model may be refined and optimized if more accurate information is available in subsequent phases of the project lifecycle. The approach itself can actually be used at any time during the project life cycle and, if possible, even after the system has been developed. In the latter case, the measured run-time parameter values can be used, instead of estimated values, to define the optimal deployment model. The runtime parameter values can be collected by using tools that collect activities (e.g. topic updates) of domain participants.

9. RELATED WORK

The allocation of software units on computing systems has applications in different computing domains such as embedded systems, local/wide area distributed systems, parallel and distributed simulations, etc. In our earlier work, we have carried out a systematic review to identify the obstacles of DDS based systems [12]. One of the identified key obstacles that we derived from the systematic review was indeed the task allocation problem. In this paper, we have provided a systematic approach with the corresponding toolset to tackle this task allocation problem.

In the literature, we can observe that some of the studies propose concrete approaches for specific domains (such as Parallel and Distributed Systems or DDS) while others provide more generic approaches that can be configured to use in different domains. In this work, we have focused on the allocation problem that is directly focused on the DDS domain. In our early work, we have provided an approach for deriving feasible deployment alternatives for parallel and distributed simulation systems in [5] and tool support for the approach in [6]. From a generic perspective, the solved allocation problem is characterized as the Multi-Processor Task Assignment (MPTA) problem [57], [36] which is a general problem that can be applied to different domains. Each MPTA problem however requires a specific approach and dedicated steps to solve the allocation problem. In this study, it is clear that the domain of DDS is specific and different and provides additional challenges including modeling the DDS system, the individual steps of the overall approach and the corresponding toolset. Our work could be further specialized by considering specific QoS parameters (such as reliability) to derive feasible alternatives. We consider this as a further complementary research.

Several generic approaches can be identified in the literature to provide a solution for the allocation problem. For example, in [14], the authors introduced a generic quality optimization framework that can be used for different component based models. Similarly, in [16] the authors propose an extensible framework that supports formal modeling of distributed software systems. The study provides a set of tailorable algorithms for finding optimized deployment architectures with respect to multiple, possibly conflicting QoS (Quality of Service) dimensions. The study also provides a visual deployment architecture modeling and analysis environment for the framework. Similar to our work, the authors evaluated the framework with simulated distributed system scenarios.

In [10], the authors use heuristics and Analytic Hierarchy Process (AHP) [23] for weighted multi-objective design space exploration. The main objective of the study is to support systems architects in complex allocation decisions in the early design phases. In [1] the authors use constructive algorithms for deployment optimization of embedded systems. Hereby, an Ant Colony Optimization (ACO) is used as a constructive multi-objective optimization strategy which is compared with a Genetic Algorithm (GA) based iterative approach. The authors conclude that they observed that constructive and iterative approaches performed similar in their experiments.

In [34], the authors define an approach to optimize the task placement and the signal to message mapping in hard-real time distributed systems domain. Authors use a mixed integer linear optimization framework to automate the assignment of priorities to tasks and messages. The optimization process aims to meet end-to-end deadline constraints and minimize latencies by leveraging worst case response time analysis. Authors validate the developed approach by applying it to an automotive system case study.

In [9], authors focused on reducing error propagation while allocating software components to distributed embedded hardware nodes. The study presents a systematic resource allocation approach for the consolidated mapping of safety critical and non-safety critical applications onto a distributed platform with consideration of dependability and real-time requirements as primary drivers. The approach focuses on finding a feasible solution satisfying multiple concurrent constraints such as ensuring criticality partitioning, avoiding error propagation and reducing interactions across components. The authors applied the approach to an actual automotive case study to prove its feasibility.

In [17], an approach is defined for automatically improving software architecture models for performance, reliability, and cost using evolutionary algorithms. Starting with a given initial architectural model, the approach iteratively modifies and evaluates architectural models by using a multi-criteria genetic algorithm based on Palladio Component Model [3]. The approach supports quantitative performance, reliability, and cost prediction of software architectures. The approach is validated by automatically investigating more than 1200 alternative design candidates for a component-based business information system and analyzing quality criteria trade-offs.

A close literature study shows thus that the task allocation is a well-known and still widely studied research area with a large application domain spreading from embedded systems to wide area distributed systems. Our approach is complementary to these approaches and is specific since it provides and integrates the necessary modeling abstractions using the extended DDS UML profile, the systematic approach, and the tool environment which can be extended for different additional functionality including different algorithms.

It should be noted that besides of the academic papers we can also identify several interesting tools on the task allocation problem, which have been provided by several vendors or which have been presented in various papers. In [2], the authors define an extensible tool for Architecture Optimization of AADL (Architecture Analysis and Description Language) Models [8] with name of ArcheOpterix. The study provides a framework to identify optimal and near optimal deployment architectures with respect to multiple quality objectives and design constraints.

The existing DDS design tools focus in general on designing the DDS application and code generation. Prismtech [18], a well-known DDS infrastructure vendor, provides Vortex OpenSplice Modeler [19] that is a domain specific model driven development tool. Vortex OpenSplice Modeler enables definition of topics (*information modeling*), DDS entities such as publishers/writers and subscribers/readers (application design). These capabilities are similar to our tool framework, but Vortex OpenSplice Modeler enables Java/C++ code generation which our tool does not provide. On the other hand, our tool framework provides an automated deployment design optimization approach which is not provided by Vortex OpenSplice Modeler. Another major DDS vendor is RTI [20], which provides UML based modeling environment for DDS. Spark Systems also provide a UML based DDS modeling

environment [21] as a plug-in of Enterprise Architect application. Another DDS vendor, MilSOFT, also provides a modeling & code generation tool for DDS [22]. All these tools providing modeling support and to some extent code generation for DDS topics and applications, but they do not provide explicit support for the deployment optimization which is the main contribution of our research. In principle, our could systematic approach could be also integrated with the existing tools.

10. CONCLUSION

An increasing number of systems are data-intensive and rely on the publish-subscribe programming model to realize the distribution aspects. The Data Distribution Service (DDS) provides a standard data-centric publish-subscribe programming model and specification for distributed systems. In addition, the OMG has provided the DDS UML Profile to support the modeling of the DDS applications. These are important developments but they do not consider the design aspects explicitly. An important design concern is of course the selection of the feasible deployment alternative given the application model, the physical resources, and the execution configurations. So far, this problem has not been explicitly addressed in the DDS literature. We have provided a systematic approach by extending the DDS UML Profile, defining the systematic approach, and a tool framework that is extensible. Furthermore, we have evaluated the approach for a relevant IoT case study on smart city engineering. The approach showed to be useful in the modeling, the design and the evaluation of the DDS deployment alternatives. The adoption of different algorithms and the ability to add new algorithms can support the system architect also in the experimentation of the different algorithms. Since in practice the task allocation problem and the selection of the feasible design alternatives is not tractable we believe that the approach and the toolset that we have provided in this paper is necessary.

In our future work, we will do research on further extension and specialization of the approach. In this context, we will consider the adoption of specific quality criteria such as reliability and further focus on the trade-off analysis using multiple quality criteria. Further, we will also consider the analysis and comparison of various algorithm implementations to further optimize the approach.

REFERENCES

- [1] A. Aleti, L. Grunske, I. Meedeniya, I. Moser, Let the ants deploy your software - An ACO based deployment optimisation strategy, in: ASE2009 - 24th IEEE/ACM Int. Conf. Autom. Softw. Eng., 2009. doi:10.1109/ASE.2009.59.
- [2] A. Aleti, S. Björnander, L. Grunske, I. Meedeniya, ArcheOpterix: An extendable tool for architecture optimization of AADL models, in: Proc. 2009 ICSE Work. Model. Methodol. Pervasive Embed. Software, MOMPES 2009, 2009: pp. 61–71. doi:10.1109/MOMPES.2009.5069138.
- [3] R. Reussner, S. Becker, E. Burger, J. Happe, M. Hauck, A. Koziolok, H. Koziolok, K. Krogmann, M. Kuperberg, The Palladio Component Model, 2011. doi:10.1016/j.jss.2008.03.066.
- [4] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, M. Paternostro, EMF: eclipse modeling framework, 2008. doi:10.1108/02641610810878585.
- [5] T. Çelik, B. Tekinerdogan, K. İmre, Deriving Feasible Deployment Alternatives for Parallel and Distributed Simulation Systems, ... Comput. Simul. (.... 23 (2013) 1–24. doi:10.1145/2499913.2499917.
- [6] T. Çelik, B. Tekinerdogan, S-IDE: A tool framework for optimizing deployment architecture of High Level Architecture based simulation systems, J. Syst. Softw. 86 (2013) 2520–2541. doi:10.1016/j.jss.2013.03.013.
- [7] C. Daly, Emfatic Language Reference. 2004. <http://www.eclipse.org/gmt/epsilon/doc/articles/emfatic/>. (Last accessed at May 2017)
- [8] P.H. Feiler, D.P. Gluch, J.J. Hudak, The Architecture Analysis and Design Language (AADL): An Introduction. Technical report, (2006) CMU/SEI-2006-TN-011.
- [9] S. Islam, R. Lindström, N. Suri, Dependability driven integration of mixed criticality SW components, in: Proc. - Ninth IEEE Int. Symp. Object-Oriented Real-Time Distrib. Comput. ISORC 2006, 2006: pp. 485–495. doi:10.1109/ISORC.2006.
- [10] I. Svogor, J. Carlson, SCALL: Software component allocator for heterogeneous embedded systems, in: ACM Int. Conf. Proceeding Ser., 2015. doi:10.1145/2797433.2797501.
- [11] D.S. Kolovos, L.M. Rose, S. Bin Abid, R.F. Paige, F.A.C. Polack, G. Botterweck, Taming EMF and GMF using model transformation, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 2010: pp. 211–225. doi:10.1007/978-3-642-16145-2_15.
- [12] O. Koksall, B. Tekinerdogan, Obstacles in Data Distribution Service Middleware: A Systematic Review, Futur. Gener. Comput. Syst. 68 (2017) 191–210. doi:10.1016/j.future.2016.09.020.
- [13] D.S. Kolovos, R.F. Paige, F.A.C. Polack, Eclipse Development Tools for Epsilon, Eclipse Model. Symp. Eclipse Summit Eur. (2006).
- [14] A. Koziolok, R. Reussner, Towards a generic quality optimisation framework for component-based system models, Proc. 14th Int. ACM Sigsoft Symp. Compon. Based Softw. Eng. (2011) 103–108. doi:10.1145/2000229.2000244.
- [15] C. Lauterbach, M.C. Lin, D. Manocha, S. Borkman, E. LaFave, G. Peele, M. Bauer, (2008). Accelerating line-of-sight computations in large OneSAF terrains with dynamic events. In Proc. IITSEC.
- [16] S. Malek, N. Medvidovic, M. Mikic-Rakic, An extensible framework for improving a distributed software system's deployment architecture, IEEE Trans. Softw. Eng. 38 (2012) 73–100. doi:10.1109/TSE.2011.3.
- [17] A. Martens, H. Koziolok, S. Becker, R. Reussner, Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms, Proc. First Jt. WOSP/SIPEW Int. Conf. Perform. Eng. (2010) 105–116. doi:10.1145/1712605.1712624.
- [18] PrismTech, <http://www.prismtech.com/>
- [19] PrismTech, Vortex OpenSplice Modeler, <http://www.prismtech.com/vortex/vortex-opensplice/tools/modeler> (Last accessed at May 2017)
- [20] RTI, <http://www.rti.com/> (Last accessed at May 2017)
- [21] Spark Systems Enterprise Architect DDS Plugin, <http://www.sparxsystems.com/products/mdg/tech/dds/> (Last accessed at May 2017)
- [22] MilSOFT, MilDDS, <http://dds.milsoft.com.tr/en/dds-home.php> (Last accessed at May 2017)
- [23] T.L. Saaty, What is the Analytic Hierarchy Process?, in: Math. Model. Decis. Support, 1988: pp. 109–121. doi:10.1007/978-3-642-83555-1_5.
- [24] B. Ucar, C. Aykanat, K. Kaya, M. İkinci, Task assignment in heterogeneous computing systems, J. Parallel Distrib. Comput. 66 (2006) 32–46. doi:10.1016/j.jpdc.2005.06.014.
- [25] M. Voelter, B. Kolb, S. Efftinge, A. Haase, From Front End to Code – MDSD in Practice, (2006), <http://www.eclipse.org/articles/Article-FromFrontendToCode-MDSDInPractice/article.html>.
- [26] OMG, Apr 2015, Data Distribution Service for Real Time Systems (DDS), v1.4, <http://www.omg.org/spec/DDS/1.4/>
- [27] OMG, Nov 2012, Common Object Request Broker Architecture (CORBA), <http://www.omg.org/spec/CORBA/> (Last accessed at May 2017)
- [28] ORACLE, March 2002, Java Message Service (JMS), <http://www.oracle.com/technetwork/java/docs-136352.html> (Last accessed at May 2017)
- [29] OMG, May 2010, UML Profile for Data Distribution Specification, Beta 1, <http://www.omg.org/cgi-bin/doc?ptc/10-05-17.pdf> (Last accessed at May 2017)
- [30] Enterprise Architect, <http://sparxsystems.com/products/ea/index.html> (Last accessed at May 2017)
- [31] Y. Yoshikawa, S. Hirasawa, M. Yamamoto, A. Sato, and M. Takahashi, “Hitachi’s Vision of the Smart City”, Hitachi Review Vol. 61, No. 3, pp. 111–118, 2012.
- [32] K. Iijima, A. Shimura, S. Hirasawa, and H. Kato, “Systems Development Technology for Public Infrastructure”, Hitachi Review Vol. 61, No. 3, pp. 159–166, 2012.
- [33] P. Kruchten, The Rational Unified Process: An Introduction, 2003. doi:10.1109/ICSE.2002.146346.
- [34] W. Zheng, Q. Zhu, M. Di Natale, A.S. Vincentelli, Definition of task allocation and priority assignment in hard real-time distributed systems, in: Proc. - Real-Time Syst. Symp., 2007: pp. 161–170. doi:10.1109/RTSS.2007.40.

Appendix A – Metamodel for OMG UML Profile for Data Distribution Specification

Fig. 1 Metamodel for DDS UML Profile/Common/Types Package

Fig. 2 Metamodel for DDS UML Profile/DCPS/Domain Package

Fig. 4 Metamodel for DDS UML Profile/Common/Topic Package

Fig. 5 Topic Definition Metamodel

Fig. 6 Metamodel for DDS UML Profile/DCPS/DCPS Package

Appendix B – Selected Deployment Alternatives

Table 10. Deployment Results for DMG_GeneticAlgorithm

Instance Name	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	TOTAL
dpiAmbulance	2			2	7	1	2	1	2		17
dpiBicycle	24	19	14	19	23	21	16	17	14	17	184
dpiBus	6	3	8	3	4	3	4	4	4	7	46
dpiCar	74	83	155	136	162	155	186	173	148	163	1435
dpiTaxi			9	9	39	10	14	14	19	10	124
dpiTruck				5	5	4	3	4	4	3	28
dpiBicycleStation				2	5	3	1	1	8	3	23
dpiCongestionSensor					16	8	10	10	9	9	62
dpiIncidentSensor					11	4	4	5	2	3	29
dpiParkingLot					8	6	4	6	4	5	33
dpiSpeedCamera					9	6	7	14	4	8	48
dpiTrafficLight		1	13		4	36	17	13	19	22	125
dpiWheatherSensor			1		2	1	1	6	1	4	16
dpiIncidentMan			2		4		5	1	2	1	15
dpiLoggerMan			3				1	4	4	3	15
dpiTicketMan				6		1	1	3	2	2	15
dpiTrafficMan				2			5	2	1	5	15
dpiParkingMan				1			8	2	2	2	15
dpiVehicleMan				5			2	2	4	2	15
dpiWhetherMan				1			6	4	1	3	15
TOTAL	106	106	205	191	299	259	297	286	254	272	2275

Table 11. Deployment Results for DMG_SequentialAllocation

Instance Name	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	TOTAL
dpiAmbulance	2	2	2	2	2	2	2	1	1	1	17
dpiBicycle	19	18	18	18	18	18	18	19	19	19	184
dpiBus	4	5	5	5	5	5	5	4	4	4	46
dpiCar	81	81	188	188	179	143	143	144	144	144	1435
dpiTaxi					63	13	12	12	12	12	124
dpiTruck					14	2	3	3	3	3	28
dpiBicycleStation					11	3	3	2	2	2	23
dpiCongestionSensor					18	18	6	7	7	6	62
dpiIncidentSensor						18	3	3	2	3	29
dpiParkingLot						19	3	3	4	4	33
dpiSpeedCamera						29	5	5	5	4	48
dpiTrafficLight						15	73	12	12	13	125
dpiWheatherSensor							10	2	2	2	16
dpiIncidentMan							12	1	1	1	15
dpiLoggerMan							9	2	2	2	15
dpiTicketMan							12	1	1	1	15
dpiTrafficMan							9	2	2	2	15
dpiParkingMan							12	1	1	1	15
dpiVehicleMan							9	2	2	2	15
dpiWhetherMan								13	1	1	15
TOTAL	106	106	213	213	310	285	349	239	227	227	2275

Table 12. Deployment Results for DMG_MinimumNodeAllocation

Instance Name	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	TOTAL
dpiAmbulance									1	16	17
dpiBicycle										184	184
dpiBus										46	46
dpiCar					320	67	426	426		196	1435
dpiTaxi						124					124
dpiTruck						28					28
dpiBicycleStation									23		23
dpiCongestionSensor									62		62
dpiIncidentSensor									29		29
dpiParkingLot									33		33
dpiSpeedCamera									48		48
dpiTrafficLight									73	52	125
dpiWheatherSensor										16	16
dpiIncidentMan									15		15
dpiLoggerMan									15		15
dpiTicketMan									15		15
dpiTrafficMan									15		15
dpiParkingMan									15		15
dpiVehicleMan									15		15
dpiWhetherMan									15		15
TOTAL	0	0	0	0	320	219	426	426	374	510	2275